File - OptimizedLinkedList.java

```java
package ch.heig.sio.lab1.groupD.Utilities;

/**
 * A simple linked list class that implements only what's useful in this lab. It also allows manual node manipulation.
 * This manual manipulation is useful for inserting in the list in O(1) complexity.
 *
 * @param <T>   The type of the value of the list
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class OptimizedLinkedList<T>{

    public static class Node<T>{
    final private T value;
        private Node<T> next;
        private Node<T> previous;

        Node(T value) {
            this.value = value;
            this.next = null;
            this.previous = null;
        }

        public T getValue() {
            return value;
        }

        public Node<T> getNext() {
            return next;
        }

        public Node<T> getPrevious(){
            return previous;
        }
    }

    private Node<T> head;
    private Node<T> tail;

    private int size;

    /**
     * Insert a value after a specified node.
     * @param n The node after which the value should be inserted
     * @param value The value to insert, it will create a new node with this value
     * @return The newly created node
     */
    public Node<T> insertAfter(Node<T> n, T value){
        Node<T> newNode = new Node<>(value);
        Node<T> afterN = n.next;
        if (afterN != null) {
            afterN.previous = newNode;
        }
        n.next = newNode;
        newNode.previous = n;
        newNode.next = afterN;

        if (n == tail) {
            tail = newNode;
        } else

        ++size;

        return newNode;
    }
```

Au milieu

Trop meta (décrivez ce qu'elle permet réellement, par exemple, comme vous l'indiquez après, un accès direct aux noeuds) et pas vraiment exact, une liste doublement chainée n'est typiquement pas utile ici (vous ne faites jamais usage de ce lien supplémentaire).

Pas utilisé

File - OptimizedLinkedList.java

```java

    /**
     * Get the head of the list
     * @return The head of the list
     */
    public Node<T> getFirst(){
        return head;
    }

    /**
     * Add a value at the end of the list
     * @param value The value to add
     * @return The newly created node
     */
    public Node<T> add(T value){
        Node<T> newNode = new Node<>(value);

        if(isEmpty()){
            tail = newNode;
            head = newNode;
        } else {
            tail.next = newNode;
            newNode.previous = tail;
            tail = newNode;
        }

        ++size;

        return newNode;
    }

    /**
     * Remove a node from the list
     * WARNING : This function is unsafe, it doesn't check if the node is in the list !
     * It is unsafe so it can be O(1) complexity
     *
     * @param node The node to remove
     */
    public void remove(Node<T> node){
        if (node == null) {
            return;
        }

        if(isEmpty()){
            return;
        }

        if (node == head) {
            head = node.next;
        }

        if (node == tail) {
            tail = node.previous;
        }
        // Tous les noeuds sauf la tête
        if (node.previous != null) {
            node.previous.next = node.next;
        }
         // Tous les noeuds sauf la queue
        if (node.next != null) {
            node.next.previous = node.previous;
        }

        --size;
    }

```

File - OptimizedLinkedList.java

```
132     public boolean isEmpty(){
133         return tail == null && head == null;
134     }
135 
136     public int size(){
137         return size;
138     }
139 
140 }
```

File - RandomInsert.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

import ch.heig.sio.lab1.display.TspHeuristicObserver;
import ch.heig.sio.lab1.groupD.Utilities.OptimizedLinkedList;
import ch.heig.sio.lab1.tsp.TspData;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

/**
 * The random heuristic used in the {@link GenericConstructiveHeuristic} to create a tour from it.
 * It contains a list of cities not visited in a random order from which it gets the city to visit
.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class RandomInsert extends GenericConstructiveHeuristic{
    ArrayList<Integer> citiesToVisit;
    Random rand = new Random();
    int citiesToVisitIndex;
private

     * @param observer          The observer used to update the GUI
     */
    @Override
    public void insertLogic(TspData data, int startCityIndex, TspHeuristicObserver observer) {
        generateShuffledCityToVisit(data, startCityIndex);

        while(citiesToVisitIndex < data.getNumberOfCities() - 1){ //nbCities - 1 since we don't
count the start city
            insertCity(getUnusedCity(),data);
            observer.update(calculateEdges());
        }
    }

    /**
     * Get a random not visited city.
     *
     * @return a random city that hasn't been visited before
     */
    private int getUnusedCity() {
        return citiesToVisit.get(citiesToVisitIndex++);
    }


    /**
     * Populate the list of cities to visit in a random order.
     *
     * @param data              The TspData to get the number of cities from
     * @param startCityIndex    The city from which the tour starts that will be skipped in the
list
     */
    private void generateShuffledCityToVisit(TspData data, int startCityIndex){
        rand.setSeed(0x134DA73);
        int nbOfCities = data.getNumberOfCities();
        citiesToVisitIndex = 0;
        citiesToVisit = new ArrayList<>(nbOfCities);

        for (int i = 0; i < nbOfCities; ++i) {
            if(i != startCityIndex){
```

Détail d'implémentation, expliquez ce que l'heuristique fait. Cela peut-être intéressant de dévoiler la structure interne d'une classe (l'exemple le plus évident étant ArrayList et LinkedList, ou pour indiquer si un algorithme est récursif afin d'éviter les stack overflows), mais généralement on évite.

Evitez de créer des attributs pour des variables qui n'ont pas de raison de survivre à l'appel de votre méthode. Ce sont basiquement des variables globales qui polluent le scope de la classe et ajoutent un état non-nécessaire à votre objet, ce qui a quelques conséquences :

1. Mémoire : elle ne peut pas être libérée entre deux appels ;
2. Maintenance/Compréhension : il faut impérativement réinitialiser les attributs correctement et il est plus facile de raisonner sans effets de bord ;
3. Concurrence : un objet qui pouvait être facilement partagé entre plusieurs threads ne le peut plus.

Méthode pas nécessaire (une ligne)

On doit pouvoir modifier la graine sans recompiler vos classes.

File - RandomInsert.java

```
64                  citiesToVisit.add(i);
65              }
66          }
67
68          Collections.shuffle(citiesToVisit, rand);
69      }
70 }
71
```

File - Gui.java

```java
package ch.heig.sio.lab1.groupD;

import ch.heig.sio.lab1.display.HeuristicComboItem;
import ch.heig.sio.lab1.display.TspSolverGui;
import ch.heig.sio.lab1.groupD.heuristics.ClosestFirstInsert;
import ch.heig.sio.lab1.groupD.heuristics.FarthestFirstInsert;
import ch.heig.sio.lab1.groupD.heuristics.RandomInsert;
import ch.heig.sio.lab1.sample.CanonicalTour;
//import com.formdev.flatlaf.FlatLightLaf;

public final class Gui {
  public static void main(String[] args) {
    HeuristicComboItem[] heuristics = {
        new HeuristicComboItem("Canonical tour", new CanonicalTour()),
            new HeuristicComboItem("Random insert", new RandomInsert()),
            new HeuristicComboItem("Closest First", new ClosestFirstInsert()),
            new HeuristicComboItem("Farthest First", new FarthestFirstInsert()),
    };

    // May not work on all platforms, comment out if necessary
    System.setProperty("sun.java2d.opengl", "true");

    //FlatLightLaf.setup();
    new TspSolverGui(1400, 800, "TSP solver", heuristics);
  }
}

```

File - GenericConstructiveHeuristic.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

import ch.heig.sio.lab1.display.ObservableTspConstructiveHeuristic;
import ch.heig.sio.lab1.display.TspHeuristicObserver;
import ch.heig.sio.lab1.groupD.Utilities.OptimizedLinkedList;
import ch.heig.sio.lab1.tsp.Edge;
import ch.heig.sio.lab1.tsp.TspData;
import ch.heig.sio.lab1.tsp.TspTour;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;


/**
 * The abstract class for the heuristic implemented. Vague
 * It contains the logic for inserting a city in the tour and updating the GUI.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public abstract class GenericConstructiveHeuristic implements ObservableTspConstructiveHeuristic
    {  Si package-private, alors la classe elle-même doit être package-private (et, au passage, vous ne pouvez pas la référencer dans votre API publique puisqu'il s'agit d'un détail d'implémentation). C'est la solution à préférer par défaut. Sinon, si la classe fait partie de l'API publique, la visibilité est protected. Cela exige souvent de fournir plus de garanties (vérification des invariants, etc.).

    OptimizedLinkedList<Integer> cycleCities;
    int distance;  Par rapport à un commentaire précédent, une bonne manière de se débarasser de ces deux attributs est ici :
                   1. De modifier insertLogic pour retourner la distance ;
                   2. De passer cycleCities en paramètre des méthodes qui l'utilisent.

    /**
     * Compute a tour using the chosen heuristic in the subclass.
     * @param data              Data of problem instance
     * @param startCityIndex    Starting city
     * @param observer          Observer to notify of the progress
     * @return                  The computed tour
     */
    @Override
    public TspTour computeTour(TspData data, int startCityIndex, TspHeuristicObserver observer) {

        distance = 0;
        int nbCities = data.getNumberOfCities();

        cycleCities = new OptimizedLinkedList<>();
        int[] finalCycle = new int[nbCities];

        //Adding the start city
        cycleCities.add(startCityIndex);

        //Call the insertLogic of the subclass
        insertLogic(data, startCityIndex, observer);

        //Populate the finalCycle array
        OptimizedLinkedList.Node<Integer> currNode = cycleCities.getFirst();
        for(int i = 0; i < nbCities; ++i){
            finalCycle[i] = currNode.getValue();
            currNode = currNode.getNext();
        }

        return new TspTour(data, finalCycle, distance);
    }

    public abstract void insertLogic(TspData data, int startCityIndex, TspHeuristicObserver
observer);

    /**
     * Calculate the edges of the tour, used to display the tour in the GUI.
     * @return An iterator over the edges of the tour
     */
```

```java
    public Iterator<Edge> calculateEdges() {
        List<Edge> res = new ArrayList<>();
        OptimizedLinkedList.Node<Integer> currentNode = cycleCities.getFirst();

        if (currentNode == null) {
            return res.iterator();
        }

        OptimizedLinkedList.Node<Integer> firstNode = currentNode;

        // Loop over the cities in the tour to create the edges
        do {
            OptimizedLinkedList.Node<Integer> nextNode = currentNode.getNext();
            if (nextNode == null) {
                nextNode = firstNode; // Loop back to the start
            }
            res.add(new Edge(currentNode.getValue(), nextNode.getValue()));
            currentNode = nextNode;

        } while (currentNode != firstNode);

        return res.iterator();
    }

    /**
     * Insert a city at the optimal position in the tour.
     * @param index the index of the city to insert
     * @param data  the TspData object containing the distances
     */
    public void insertCity(int index, TspData data) {

        // Find the shortest distance between two already existing vertices
        int bestDistance = Integer.MAX_VALUE;
        OptimizedLinkedList.Node<Integer> bestNode = null;

        OptimizedLinkedList.Node<Integer> currentNode = cycleCities.getFirst();
        if (currentNode == null) {
            return; // No city in the tour yet
        }

        OptimizedLinkedList.Node<Integer> firstNode = currentNode;
        do {

            OptimizedLinkedList.Node<Integer> nextNode = currentNode.getNext();
            if (nextNode == null) {
                nextNode = firstNode; // Loop back to the start
            }

            int calculatedDist = data.getDistance(currentNode.getValue(), index) +
                    data.getDistance(index, nextNode.getValue()) -
                    data.getDistance(currentNode.getValue(), nextNode.getValue());

            // If the distance is better, update the best distance and node
            if (calculatedDist < bestDistance) {
                bestDistance = calculatedDist;
                bestNode = currentNode;
            }

            currentNode = nextNode;
        } while (currentNode != firstNode);

        // The best distance has been found, insert the city now
        if (bestNode != null) {
            cycleCities.insertAfter(bestNode, index);
            distance += bestDistance;
        }
```

Il est préférable d'écrire un itérateur qui génère les arêtes à la volée que de pré-calculer toutes les arêtes, il se peut qu'aucune ne soit jamais créée. Mais il s'agit de l'interface graphique => aucune importance pour ce labo.

?

Et si la meilleure position n'a pas été trouvée il se passe quoi ? :)

```
131     }
132
133 }
134
```

File - DistanceBasedInsert.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

import ch.heig.sio.lab1.display.TspHeuristicObserver;
import ch.heig.sio.lab1.groupD.Utilities.OptimizedLinkedList;
import ch.heig.sio.lab1.tsp.TspData;

/**
 * The abstract class used in {@link GenericConstructiveHeuristic} to implement a distance based heuristic (far or close).
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public abstract class DistanceBasedInsert extends GenericConstructiveHeuristic {
    OptimizedLinkedList<CityDistancePair> outsideCycleCitiesDistance;

    /**
     * The insert logic for the distance based heuristic.
     * It starts by populating the list of cities outside the tour and then loop on the logic to add them to the tour until all are added.
     * @param data              The TspData used to get information on the cities
     * @param startCityIndex    The city from which the tour starts
     * @param observer          The observer used to update the GUI
     */
    @Override
    public void insertLogic(TspData data, int startCityIndex, TspHeuristicObserver observer) {
        outsideCycleCitiesDistance = new OptimizedLinkedList<>();

        int selectedDistance = getMinOrMax();
        OptimizedLinkedList.Node<CityDistancePair> selectedCityNode = null;

        //Populate with distance to startCityIndex and the city
        for (int i = 0; i < data.getNumberOfCities(); ++i){
            if(i == startCityIndex) continue;

            int distance = data.getDistance(i,startCityIndex);
            OptimizedLinkedList.Node<CityDistancePair> currNode = outsideCycleCitiesDistance.add(new CityDistancePair(i,distance));

            if (cityDistanceSelection(distance, selectedDistance)){
                selectedDistance = distance;
                selectedCityNode = currNode;
            }
        }

        if(selectedCityNode == null){
            System.out.println("No closest city, make sure you have at least 2 cities");
            return;
        }

        //Main loop logic
        do {
            //Add the selected city
            insertCity(selectedCityNode.getValue().getIndex(),data);
            observer.update(calculateEdges());

            //Remove the city
            outsideCycleCitiesDistance.remove(selectedCityNode);

            //Update the distances and get the new city
            selectedCityNode = updateDistanceAndGetCity(selectedCityNode.getValue(),data);

        } while (!outsideCycleCitiesDistance.isEmpty() && selectedCityNode != null);
    }

    /**
```

et sélection de la ville la plus proche de la ville de départ

Probablement un des deux en trop, dans tous les cas, transformez cette boucle en un while, cela permettra de simplifier l'initialisation en mettant toutes les valeurs à +inf et d'éviter d'appeler updateDistanceAndGetCity alors qu'il n'y a plus de villes à insérer.

File - DistanceBasedInsert.java

```java
     * Update the distance of all the cities not in the cycle to the cityToCompareTo and return the new city to add.
     * @param cityToCompareTo   The city to compare the distances to, it should be the latest city added to the cycle
     * @param data              The TspData used to get the distances
     * @return                  The new city to then add to the cycle
     */
    private OptimizedLinkedList.Node<CityDistancePair> updateDistanceAndGetCity(CityDistancePair cityToCompareTo, TspData data){
        OptimizedLinkedList.Node<CityDistancePair> currNode = outsideCycleCitiesDistance.getFirst();
        if(currNode == null) return null;

        //Update the distances
        do {
            int currDistance = currNode.getValue().getDistance();
            int maybeNextDistance = data.getDistance(cityToCompareTo.getIndex(),
                    currNode.getValue().getIndex());

            if(maybeNextDistance < currDistance){
                currNode.getValue().setDistance(maybeNextDistance);
            }

            currNode = currNode.getNext();
        } while (currNode != null);

        //Find the city based on cityDistanceSelection() implementation
        OptimizedLinkedList.Node<CityDistancePair> selectedCity = outsideCycleCitiesDistance.getFirst();
        currNode = outsideCycleCitiesDistance.getFirst();
        do {

            if(cityDistanceSelection(currNode.getValue().getDistance(),selectedCity.getValue().getDistance())){
                selectedCity = currNode;
            }

            currNode = currNode.getNext();
        } while (currNode != null);

        return selectedCity;
    }

    /**
     * The abstract method called to select the city to insert in our tour.
     * @param d1    Distance of the new city to test if selected
     * @param d2    Distance of the city already selected
     * @return      A boolean that update replaces the selected city by the tested one if True
     */
    abstract boolean cityDistanceSelection(int d1, int d2);

    /**
     * The abstract method called when initialising the selected distance at the start, useful when using a distance based heuristic.
     * @return  The initial value of the slected distance, here Integer.MAX_VALUE or Integer.MIN_VALUE
     */
    abstract int getMinOrMax();
}

```

File - CityDistancePair.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

import ch.heig.sio.lab1.groupD.Utilities.Pair;

/**
 * The specification of {@link Pair} for one city and a distance.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class CityDistancePair extends Pair<Integer,Integer> {

    public CityDistancePair(int index, int distance) {
        super(index, distance);
    }

    public int getIndex(){
        return super.getFirst();
    }

    public int getDistance(){
        return super.getSecond();
    }

    public void setDistance(int distance){
        super.setSecond(distance);
    }

}

```

Pas nécessaire d'avoir une abstraction générique pour une seule implémentation très spécifique. Utilisez directement des attributs avec des noms évocateurs.

File - AnalyzeOutput.txt

```
Statistical analysis of the results of the TSP solver for the different datasets and insertion strategies.
Group D - Lab 1 - TSP

The performance is a percentage of the mean distance compared to the optimal distance. It should never be below 100% and the closer to 100% the better.

Analysis for dataset: pcb442
Optimal tour length: 50'778
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
Max                 59'650.00         61'829.00         59'875.00
Min                 56'983.00         59'212.00         55'468.00
Median              58'324.00         60'287.00         57'398.00
Mean                58'263.21         60'345.02         57'407.23
Standard Deviation  494.66            543.54            723.77
Performance (%)     114.74            118.84            113.06

Analysis for dataset: att532
Optimal tour length: 86'729
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
Max                 98'426.00         108'741.00        97'886.00
Min                 94'465.00         106'471.00        91'483.00
Median              96'823.00         107'443.00        94'500.50
Mean                96'783.72         107'480.69        94'532.50
Standard Deviation  694.98            420.99            1'096.84
Performance (%)     111.59            123.93            109.00

Analysis for dataset: u574
Optimal tour length: 36'905
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
Max                 42'099.00         46'162.00         42'027.00
Min                 40'787.00         44'731.00         39'379.00
Median              41'411.00         45'559.00         40'819.00
Mean                41'403.64         45'518.51         40'804.96
Standard Deviation  306.59            313.00            456.40
Performance (%)     112.19            123.34            110.57

Analysis for dataset: pcb1173
Optimal tour length: 56'892
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
Max                 67'123.00         72'806.00         67'238.00
Min                 65'199.00         70'768.00         63'628.00
Median              65'997.00         72'358.00         65'704.00
Mean                66'036.55         72'216.36         65'705.78
Standard Deviation  348.88            446.77            499.12
Performance (%)     116.07            126.94            115.49

Analysis for dataset: nrw1379
Optimal tour length: 56'638
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
Max                 64'375.00         69'778.00         64'229.00
Min                 62'769.00         68'982.00         61'518.00
Median              63'667.00         69'473.00         62'870.00
Mean                63'659.52         69'459.90         62'879.20
Standard Deviation  251.58            143.23            397.71
Performance (%)     112.40            122.64            111.02

Analysis for dataset: u1817
Optimal tour length: 57'201
Metric              Random insert     Closest First     Farthest First
-------------------------------------------------------------------------
```

Performance par rapport à quel indicateur ? Il faudrait également l'afficher pour tous les indicateurs.

65	Max	69’518.00	71’537.00	71’227.00
66	Min	67’620.00	70’139.00	67’089.00
67	Median	68’678.00	70’931.00	69’019.00
68	Mean	68’644.25	70’906.58	69’001.62
69	Standard Deviation	392.32	238.36	553.75
70	Performance (%)	120.01	123.96	120.63

File - FarthestFirstInsert.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

/**
 * The specification for {@link DistanceBasedInsert} for the farthest first heuristic.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class FarthestFirstInsert extends DistanceBasedInsert{
    @Override
    boolean cityDistanceSelection(int d1, int d2) {
        return d1 > d2;
    }

    @Override
    int getMinOrMax() {
        return Integer.MIN_VALUE;
    }
}

```

File - Pair.java

```java
package ch.heig.sio.lab1.groupD.Utilities;


/**
 * A simple pair class that allows to only have basics methods.
 * @param <T>   The type of the first value of the pair
 * @param <U>   The type of the second value of the pair
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class Pair<T,U>{
    private T firstValue;
    private U secondValue;

    public Pair(T v1, U v2){
        firstValue = v1;
        secondValue = v2;
    }

    public T getFirst() {
        return firstValue;
    }

    public U getSecond(){
        return secondValue;
    }

    public void setFirst(T val){
        firstValue = val;
    }

    public void setSecond(U val){
        secondValue = val;
    }
}

```

File - ClosestFirstInsert.java

```java
package ch.heig.sio.lab1.groupD.heuristics;

/**
 * The specification of {@link DistanceBasedInsert} for the closest first heuristic.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public class ClosestFirstInsert extends DistanceBasedInsert {

    @Override
    boolean cityDistanceSelection(int d1, int d2) {
        return d1 < d2;
    }

    @Override
    int getMinOrMax() {
        return Integer.MAX_VALUE;
    }
}

```

File - Analyze.java

```java
package ch.heig.sio.lab1.groupD;

import ch.heig.sio.lab1.display.HeuristicComboItem;
import ch.heig.sio.lab1.groupD.heuristics.ClosestFirstInsert;
import ch.heig.sio.lab1.groupD.heuristics.FarthestFirstInsert;
import ch.heig.sio.lab1.groupD.heuristics.RandomInsert;
import ch.heig.sio.lab1.tsp.TspData;

import java.text.DecimalFormat;
import java.util.*;

import static java.util.Collections.max;
import static java.util.Collections.min;

/**
 * Used to get different metrics on the heuristics.
 * @author Edwin Häffner
 * @author Arthur Junod
 */
public final class Analyze {

    private record Statistics(double max, double min, double median, double mean, double stdDev
) {}

    public static void main(String[] args) {

        //Array of all the heuristics
        HeuristicComboItem[] heuristics = {
                new HeuristicComboItem("Random insert", new RandomInsert()),
                new HeuristicComboItem("Closest First", new ClosestFirstInsert()),
                new HeuristicComboItem("Farthest First", new FarthestFirstInsert()),
        };

        // Array of files
        String[] files = {"pcb442", "att532", "u574", "pcb1173", "nrw1379", "u1817"};
        long[] optimalDistances = {50778, 86729, 36905, 56892, 56638, 57201};

        System.out.println("Analyzing heuristics...");
        System.out.println("The performance is a percentage of the mean distance compared to the
optimal distance. It should never be below 100% and the closer to 100% the better.");

        // Loop through all the files
        for (int fileIndex = 0; fileIndex < files.length; fileIndex++) {
            String file = files[fileIndex];

            // Open the file and analyze the heuristics
            try {
                TspData data = TspData.fromFile("data/" + file + ".dat");
                int numberOfCities = data.getNumberOfCities();

                System.out.println("\nProcessing dataset: " + file + ".dat (" + numberOfCities +
" cities)");
                Map<String, Statistics> statsMap = new LinkedHashMap<>();

                // Loop through all the heuristics
                for (HeuristicComboItem heuristic : heuristics) {
                    ArrayList<Long> results = new ArrayList<>();
                    long meanValue = 0;

                    // Loop through all the cities
                    for (int i = 0; i < numberOfCities; ++i) {
                        long length = heuristic.computeTour(data, i).length();
                        results.add(length);
                        meanValue += length;

                        updateProgress(i + 1, numberOfCities, heuristic.toString());
```

File - Analyze.java

```
                    }

                    double mean = (double) meanValue / data.getNumberOfCities();
                    double medianValue = median(results);
                    double stdDevValue = stdDev(results, mean);

                    statsMap.put(heuristic.toString(), new Statistics(
                            max(results),
                            min(results),
                            medianValue,
                            mean,
                            stdDevValue
                    ));
                }

                printStatistics(file, statsMap, optimalDistances[fileIndex]);

            } catch (Exception e) {
                System.err.println("There was an error in processing " + file + ".dat");
                System.err.println(e.getMessage());
                return;
            }
        }
    }

    public static double median(List<Long> values) {
        Collections.sort(values);
        int middle = values.size() / 2;
        if (values.size() % 2 == 0) {
            return (values.get(middle - 1) + values.get(middle)) / 2.0;
        } else {
            return values.get(middle);
        }
    }

    public static double stdDev(List<Long> values, double mean) {
        double sum = 0;
        for (Long value : values) {
            sum += Math.pow(value - mean, 2);
        }
        return Math.sqrt(sum / (values.size() - 1));
    }

    /**
     * Print the statistics for the heuristics in a nice readable format.
     * Generated with ClaudeAI.
     *
     * @param filename The name of the file being analyzed
     * @param heuristicStats The statistics for each heuristic
     * @param optimalDistance The optimal distance for the dataset
     */
    private static void printStatistics(String filename, Map<String, Statistics> heuristicStats
    , long optimalDistance) {
        int metricWidth = 20;
        int valueWidth = 18;

        // Print header
        System.out.println("\nAnalysis for dataset: " + filename);
        System.out.println("Optimal tour length: " + String.format("%,d", optimalDistance));
        System.out.printf("%-" + metricWidth + "s", "Metric");
        for (String heuristic : heuristicStats.keySet()) {
            System.out.printf("%-" + valueWidth + "s", heuristic);
        }
        System.out.println();

        // Print separator
```

```
129         System.out.println("-".repeat(metricWidth + (valueWidth * heuristicStats.size())));
130
131         // Print statistics
132         String[] metrics = {"Max", "Min", "Median", "Mean", "Standard Deviation", "Performance
    (%)"};
133         DecimalFormat df = new DecimalFormat("#,##0.00");
134
135         for (String metric : metrics) {
136             System.out.printf("%-" + metricWidth + "s", metric);
137             for (Statistics stats : heuristicStats.values()) {
138                 double value = switch (metric) {
139                     case "Max" -> stats.max;
140                     case "Min" -> stats.min;
141                     case "Median" -> stats.median;
142                     case "Mean" -> stats.mean;
143                     case "Standard Deviation" -> stats.stdDev;
144                     case "Performance (%)" -> (stats.mean / optimalDistance) * 100;
145                     default -> 0.0;
146                 };
147                 System.out.printf("%-" + valueWidth + "s", df.format(value));
148             }
149             System.out.println();
150         }
151     }
152
153     /**
154      * Display a progress bar for the current heuristic tested.
155      * Generated with ClaudeAI.
156      * @param current           The current city processed, used to track progress
157      * @param total             The total number of cities
158      * @param currentHeuristic  The name of the current heuristic tested
159      */
160     private static void updateProgress(int current, int total, String currentHeuristic) {
161         int progressBarWidth = 40;
162         double percentage = (double) current / total * 100;
163         int completedWidth = progressBarWidth * current / total;
164
165         // Create the progress bar
166         StringBuilder progressBar = new StringBuilder("[");
167         for (int i = 0; i < progressBarWidth; i++) {
168             if (i < completedWidth) {
169                 progressBar.append("=");
170             } else if (i == completedWidth) {
171                 progressBar.append(">");
172             } else {
173                 progressBar.append(" ");
174             }
175         }
176         progressBar.append("]");
177
178         // Print the progress bar and percentage
179         System.out.print("\r" + currentHeuristic + " Progress: " + progressBar + " " +
180                 String.format("%.1f%%", percentage) + "    "); // Extra spaces to clear any
    previous longer text
181
182         // Print newline if we're done processing
183         if (current == total) {
184             System.out.println();
185         }
186     }
187
188
189
190 }
191
192
```

Mouais

Très cool